МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
НАЦІОНАЛЬНИЙ АВІАЦІЙНИЙ УНІВЕРСИТЕТ

Кафедра комп'ютеризованих систем управління

ДОПУСТИТИ ДО ЗАХИСТУ
Завідувач кафедри

_____ Литвиненко О.Є.

«___» _____2021 р.

ДИПЛОМНИЙ ПРОЕКТ

(ПОЯСНЮВАЛЬНА ЗАПИСКА)

ДОБУВАЧА ОСВІТНЬОГО СТУПЕНЯ
"БАКАЛАВР"

Тема: Мобільний додаток моніторингу розкладу руху залізничного транспорту

Виконавець: Макарьєв Єгор Олександрович

Керівник: Халімон Наталія Федорівна

Нормоконтролер: Тупота Євгеній Вікторович

Київ 2021

НАЦІОНАЛЬНИЙ АВІАЦІЙНИЙ УНІВЕРСИТЕТ

Факультет Кібербезпеки, комп'ютерної та програмної інженерії
Кафедра комп’ютеризованих систем управління
Спеціальність 123 «Комп'ютерна інженерія»
(шифр, найменування)

Освітньо-професійна програма «Системне програмування»
Форма навчання денна

ЗАТВЕРДЖУЮ
Завідувач кафедри
_____
«_____»_____2021 р.

ЗАВДАННЯ
на виконання дипломної роботи (проєкту)

Макарьєва Єгора Олександровича
(прізвище, ім'я, по батькові випускника в родовому відмінку)

1. Тема проекту (роботи): Мобільний додаток моніторингу розкладу руху залізничного транспорту
затверджена наказом ректора від 04.02.2021 №135/ст
2. Термін виконання проекту (роботи): з 17 травня 2021 року до 20 червня 2021 року
3. Вихідні дані до проекту (роботи):
Середовище розробки *QtCreator*, мова програмування *C++*, фреймворк *Qt*, декларативна мова програмування *QML*.
4. Зміст пояснювальної записки (перелік питань, що підлягають розробці):
1) Огляд програмних засобів моніторингу розкладу руху залізничного Транспорту;
2) Проектування мобільного додатку моніторингу розкладу руху залізничного транспорту;
3) Розробка мобільного додатку моніторингу розкладу руху залізничного транспорту.
5. Перелік обов’язкового графічного матеріалу:
1) Схема алгоритму роботи програмного засобу моніторингу розкладу руху залізничного транспорту;
2) Діаграма класів розроблюваного проекту;
3) Вікно сторінки пошуку рейсу за його номером;
4) Вікно сторінки пошуку рейсів за напрямком;
5) Вікно сторінки пошуку рейсів за станцією.

6. Календарний план

№ п/п	Етапи виконання дипломного проекту	Термін виконання етапів	Примітка
1	Ознайомитись з постановкою задачі дипломного проектування	17.05.21 – 18.05.21	
2	Вивчити спеціальну літературу і технічну документацію	18.05.21 – 19.05.21	
3	Проаналізувати програмні засоби моніторингу розкладу руху залізничного транспорту	19.05.21 – 20.05.21	
4	Написати розділ 1.	20.05.21 – 21.05.21	
5	Проаналізувати обрану платформу розробки. Спроектувати програмний засіб моніторингу розкладу руху залізничного транспорту.	21.05.21 – 23.05.21	
6	Написати розділ 2.	23.05.21 – 26.05.21	
7	Виконати розробку програмного засобу моніторингу розкладу руху залізничного транспорту	26.05.21 – 18.05.21	
8	Написати розділ 3.	28.05.21 – 07.06.21	
9	Оформити пояснювальну записку. Підготовити графічний демонстраційний матеріал	07.06.21 – 14.06.21	
10	Підготовити презентацію, доклад	14.06.21 – 17.06.21	

7. Дата видачі завдання 18.05.21

Керівник дипломного проекту _____ Халімон Наталія Федорівна
(підпис)

Завдання прийняв до виконання _____ Макарьєв Єгор Олександрович
(підпис випускника) (П.І.Б.)

РЕФЕРАТ

Пояснювальна записка до дипломного проекту «Мобільний додаток моніторингу розкладу руху залізничного транспорту»: 54 с., 9 рис., 0 таблиці, 11 літературних джерел, 1 додаток.

МОБІЛЬНИЙ ДОДАТОК, КРОС-ПЛАТФОРМОВІ ЗАСОБИ РОЗРОБКИ МОБІЛЬНИХ ДОДАТКІВ, РОЗКЛАД РУХУ, ЗАЛІЗНИЧНИЙ ТРАНСПОРТ, МОВА ПРОГРАМУВАННЯ C++, ФРЕЙМВОРК QT

Об’єкт дослідження дипломної роботи – програмні засоби для моніторингу розкладу руху залізничного транспорту.

Предмет дослідження дипломної роботи – проектування та розробка мобільного програмного засобу для моніторингу розкладу руху залізничного транспорту.

Мета дослідження – спроектувати мобільний програмний засіб для моніторингу розкладу руху залізничного транспорту.

Розроблений програмний засіб, що був створений під час виконання дипломного проектування рекомендується використовувати для моніторингу розкладу руху приміського залізничного транспорту у м. Київ та Київської області.

Дипломний проект присвячений актуальній тематиці розробки крос-платформових мобільних додатків для операційних систем *Android* та *iOS*.

ЗМІСТ

ВСТУП

На сьогоднішній день існують два типи додатків моніторингу розкладу руху залізничного транспорту: для стаціонарних платформ та мобільних. Кожен із них має різне призначення та свої особливості. Для визначення цих особливостей потрібно детальніше розглянути вже існуючі сервіси та програмні додатки.

Для стаціонарних платформ можна виділити два найпопулярніші сервіси для моніторингу розкладу руху залізничного транспорту в Україні: сайт «*poezdato.net*» та офіційний сайт Укрзалізниці – «*uz.gov.ua*». Вони зручні у використанні на персональних комп'ютерах, що використовують такі операційні системи, як *Windows, Linux, MacOS* тощо. Подібні сервіси, зазвичай, надають користувачеві невеликий спектр послуг, а саме: пошук рейсів за напрямком, пошук рейсу за його номером та пошук рейсу по станції. Але, їхня особливість полягає у тому, що майже всі вони надають послугу покупки, або бронювання квитків на потяги, що є далеко не у всіх мобільних додатках.

На ринку мобільних додатків існує багато програмних засобів для моніторингу розкладу руху залізничного транспорту, тому розглянемо чотири програми: мобільний додаток «Моя Електричка», мобільний додаток «Розклади приміських поїздів України - *UZTrains*», мобільний додаток «Яндекс.Электрички», мобільний додаток «Укрзалізниця». Вони зручні у використанні на мобільних пристроях, що використовують операційні системи *Android* або *iOS*. Подібні додатки, зазвичай, надають ширший спектр послуг, аніж аналоги для стаціонарних платформ, а саме: пошук рейсів за напрямком, пошук рейсу за його номером, пошук рейсу по станції, збереження потрібних рейсів для їх перегляду без підключення до Інтернету. Деякі з них пропонують навіть наступні функції: перегляд маршруту рейсу на мапі, або додання вікна на головний екран, що містить важливі рейси та їх відправлення від певної станції. Їхня головна особливість полягає у тому, що користувач має можливість завжди переглянути розклад руху потрібного потягу за допомогою мобільного

пристрою, навіть не перебуваючи вдома. Проте, на відміну від стаціонарних платформ, не всі подібні мобільні додатки надають можливість купування та бронювання квитків на залізничний транспорт.

При розробці дипломного проекту було обрано мову програмування *C++* із фреймворком *Qt*. Розробка велася в середовищі *QtQuick* із набором інструментів для розробки додатків для мобільних платформ *Android* – *AndroidSDK*. Для розробки графічного інтерфейсу було обрано мову програмування *QML*.

Мова програмування *C++* була обрана через те, що вона має всі необхідні інструменти для управління оперативної пам'яттю пристрою, що пришвидшує роботу додатку. Прикладом таких інструментів є покажчики. Вони дозволяють зберігати адресу певного блоку оперативної пам'яті. Якщо покажчик вказує на адресу певної змінної, то є можливість за допомогою нього змінювати значення цієї змінної з однієї частині коду, у іншій, що допомагає уникнути створення додаткових змінних для виконання даної операції.

Фреймворк *Qt* був обраний через те, що його бібліотеки додають інструменти, що полегшують розробку програмного засобу. До таких інструментів можна віднести різноманітні контейнери та алгоритми, що відсутні в *C++*.

Середовище *QtQuick* було обрано через те, що воно надає інструменти для роботи із засобом розробки мобільних додатків *AndroidSDK*. Також воно підтримує більшість компіляторів для мови програмування *C++*. Крім того, *QtQuick* має зручні інструменти відладки для програмних засобів на мобільних платформах, а саме інструменти управління емуляторами на базі *Android,* що дозволяє запускати програму на стаціонарних платформах.

Мова програмування графічного інтерфейсу *QML* була обрана через те, що вона надає зручні інструменти для розробки користувацького інтерфейсу для мобільних пристроїв із сенсорним управлінням. Також вона дозволяє використовувати мову програмування *JavaScript* для надання додатку більш гнучкого вигляду.

Об’єкт дослідження дипломного проекту – програмні засоби для моніторингу розкладу руху залізничного транспорту.

Предмет дослідження дипломного проекту – проектування та розробка мобільного програмного засобу для моніторингу розкладу руху залізничного транспорту.

Мета дослідження – спроектувати мобільний програмний засіб для моніторингу розкладу руху залізничного транспорту.

На даний час в нашій державі електронні табло з розкладом руху на зупинках транспорту з часом прибуття взагалі, а приміського зокрема, не дуже популярні, тому мобільний додаток, який вирішує цю проблему, є корисним, а тема дипломного проекту «Мобільний додаток моніторингу розкладу руху залізничного транспорту» є актуальною.

РОЗДІЛ 1
ОГЛЯД ПРОГРАМНИХ ЗАСОБІВ МОНІТОРИНГУ РОЗКЛАДУ РУХУ ЗАЛІЗНИЧНОГО ТРАНСПОРТУ

1.1. Програмні засоби моніторингу розкладу руху залізничного транспорту

На сьогоднішній день існують два типи додатків моніторингу залізничного транспорту: для стаціонарних платформ та мобільних. У програмах кожного типу є свої особливості як при розробці, так і при використанні. В галузі автоматизації моніторингу розкладу як приміського, так і поїздів далекого сполучення для стаціонарних платформ існують наступні сервіси:

– сайт для перегляду розкладу залізничного транспорту «*Poezdato.net*»;

– офіційний сайт Укрзалізниці «*uz.gov.ua*».

Сайт для перегляду розкладу залізничного транспорту «*Poezdato.net*». Він має наступні функції: перегляд актуального розкладу рейсів, що проходять через певну станцію; перегляд актуального розкладу рейсів, що слідують від початкової до кінцевої станції, введених користувачем; можливість пошуку рейсів по даті; можливість обирати тип потягу; можливість покупки білету на залізничний транспорт. Даний сайт має приємний та простий інтерфейс.

Офіційний сайт Укрзалізниці «*uz.gov.ua*» надає користувачеві такий набір функцій: перегляд розкладу руху приміських потягів за обраним напрямком; можливість пошуку маршруту від однієї станції до іншої з пересадкою; перегляд розкладу руху приміських потягів між станціями, або регіонами; перегляд розкладу руху по станції, або регіону; можливість пошуку рейсів по даті; покупка білетів. На жаль, дизайн та інтерфейс даного сайту вже застарілий, тому

він може викликати певні труднощі у користуванні для звичайного відвідувача.

В галузі автоматизації моніторингу розкладу залізничного транспорту для мобільних платформ існують наступні сервіси:

– мобільний додаток «Моя Електричка»;

– мобільний додаток «Розклади приміських поїздів України - *UZTrains*»;

– мобільний додаток «Яндекс.Электрички»;

– мобільний додаток «Укрзалізниця».

Додаток «Моя Електричка» від розробника «*Red Beam*», створений для моніторингу розкладу приміського залізничного транспорту по всій Україні. Дана програма має три основні сторінки: перегляд рейсів, що прямують від однієї станції до іншої; перегляд розкладу приміських поїздів, що проходять через певну станцію; збереженні маршрути для їх перегляду без підключення до Інтернету. Біля деяких рейсів програма виводить зміни у розкладі, деякі помітки або відміну потягу.

Мобільний додаток «Розклади приміських поїздів України - *UZTrains*» є не дуже популярною програму. Він надає користувачеві наступні функції: перегляд актуального розкладу руху приміського залізничного транспорту; перегляд попередньо завантажених розкладів без Інтернету; пошук маршруту з пересадками; відображення відстаней між станціями; відображення цін квитків для приміського сполучення; сповіщення про зміни для обраних поїздів. Проте головною функцією, що відрізняє даний додаток від інших, є перегляд маршрутів по карті.

На ринку мобільних додатків є доволі популярна програма «Яндекс.Электрички», що розроблена для моніторингу розкладу руху приміського залізничного транспорту на території Російської Федерації. Програма має наступні функції: перегляд актуального розкладу руху приміського залізничного транспорту; зберігання обраних маршрутів; отримання повідомлень щодо затримок, або переносу рейсу на інший час або дату; додання на головний екран мобільного телефону вікна, що відображає розклад потрібних користувачеві рейсів; заведення будильнику, щоб не спізнитися на свій рейс. Програма має приємний та зрозумілий інтерфейс. Проте

даний додаток не функціонує на території України, тому для її жителів він є некорисним.

Мобільний додаток «Укрзалізниця» від компанії «*HBB Software*» спеціалізується не лише на приміському сполученні, але й на усіх пасажирських рейсах. Ця програма дозволяє переглянути розклад від однієї станції до іншої, та переглянути час відправлення та прибуття згідно розкладу. Також у ньому присутня функція перегляду відправлення потягів від певної станції, а також їхнє направлення. Ще є можливість побачити через які станції проходить той чи інший рейс.

1.2. Крос-платформові засоби розробки програмного забезпечення для операційних систем *Android* та *iOS*

Мобільні додатки в основному розроблюються для найпопулярніший платформ *Android* та *iOS*.Для розробки крос-платформового додатку для даних операційних систем існують наступні засоби:

– фреймворк *Qt*;

– фреймворк *Xamarin*;

– фреймворк *Ionic*.

Qt – крос-платформовий фреймворк для розробки програмного забезпечення на мові програмування *C++*. За допомогою нього розробник з легкістю може перенести свій додаток з однієї платформи на іншу, наприклад з операційної системи *Windows* на *Android* [1]. Залишається тільки розробити інтерфейс для своєї програми під те чи інше середовище.

Крім цього *Qt* має наступні переваги:

– зручна міжпроцесна взаємодія за допомогою сигналів та слотів. Сигнал викликається коли трапляється певна подія. Слот – це функція, що викликається у відповідь на певний сигнал [2];

– дана бібліотека містить в собі різноманітні контейнери(Контейнери – клас, структура даних, або абстрактний тип даних, який дозволяє створювати

колекції інших об’єктів [3]) та алгоритми, що відсутні в *C++* і полегшують процес розробки додатку;

– *Qt* активно розвивається. Навіть сьогодні фреймворк періодично оновлюється;

– зручне середовище розробки;

– детальна та впорядкована документація на офіційному сайті фреймворку, що допоможе програмісту розібратися у функціях, класах та їх методах при розробці програмного забезпечення.

Проте, не дивлячись на всі переваги даного продукту, він має свої недоліки:

– великий розмір додатків за рахунок бібліотек. Саме цей недолік може стати основним при розробці мобільного додатку, тому потрібно постійно аналізувати які бібліотеки можна замінити додатковим кодом у програмі;

– складно знайти рішення для нестандартних ситуацій.

Перевагою розробки програмних додатків на фреймворку *Qt* є те, що він надає усі інструменти для обробки великої кількості даних, бо мова програмування *C++* надає інструменти для максимально ефективного управління оперативною пам’яттю за допомогою покажчиків, що значно прискорює виконання певних алгоритмів.

Xamarin – крос-платформовий фреймворк для розробки мобільних додатків на мові програмування *C#* з використанням *.NET*. Розробка ведеться в середовищі *Visual Studio* із доволі зручним інтерфейсом.

Основні переваги *Xamarin*:

– можливість створення інтерфейсу додатку подібного до «рідного», тобто розробник зможе підключити до проекту «рідні» бібліотеки тієї чи іншої мобільної операційної системи;

– оптимальні умови тестування. Для розробників є можливість відлагодити свої проекти на хмарній платформі «*Test Cloud*», що дозволяє емулювати більше 2000 мобільних приладів, не завантажуючи їх образи на комп’ютер. Проте, ця функція платна;

– ідеальна сумісність із різними функціями мобільного телефону. Наприклад, додаток може отримувати наступні дані зі смартфону: геолокація, положення гіроскопа, акселерометра тощо;

– якісна документація.

Основні недоліки середовища:

– розмір готових проектів завеликий для мобільного додатку;

– замала продуктивність інтерактивних додатків зі складною анімацією. Проте це проблема не самого фреймворку, а мови програмування *JavaScript*, що використовується для створення графічного інтерфейсу програми на базі *Xamarin* [4].

Фреймворк *Xamarin* частіше використовують для розробки додатків, що потребують використання рідних бібліотек платформ *Android* або *iOS* та створення інтерфейсу, що буде виглядати «рідним» на будь-якій мобільній операційній системі.

Ionic – крос-платформовий фреймворк для розробки мобільних додатків на мові програмування *Java*. Це технологія, що дозволяє розробляти повноцінний додаток для *iOS* та *Android* [5]. Принцип роботи мобільних додатків на фреймворку *Ionic* зображений на рис. 1.1.

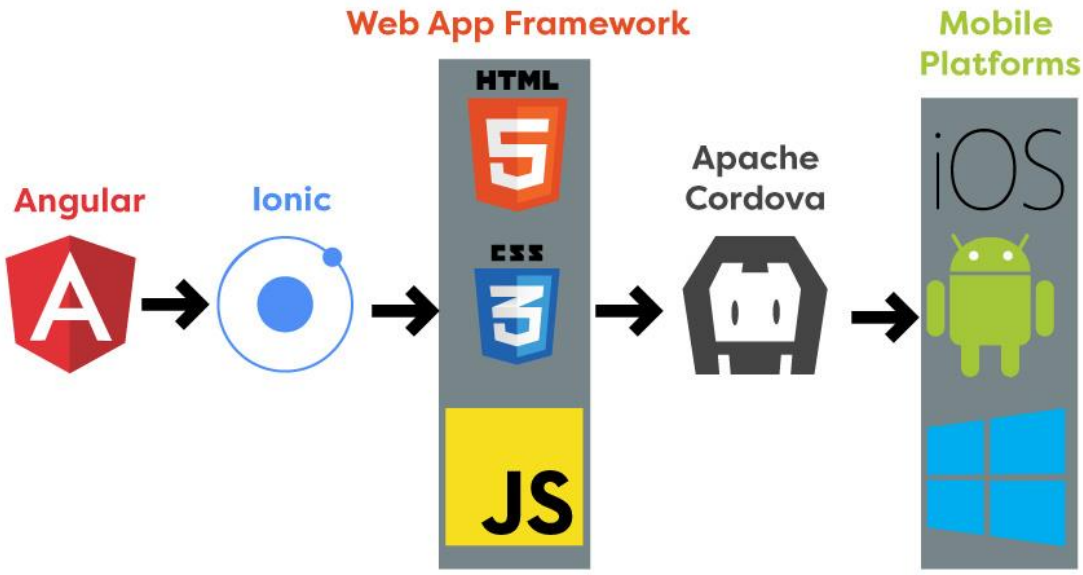

Рис. 1.1. Принцип роботи мобільних додатків на фреймворку *Ionic*

Web App Framework – це фреймворк, що призначений для розробки динамічних веб-сайтів, мережевих додатків, сервісів або ресурсів. Він спрощує доступ до бази даних та розробку інтерфейсу [6].

Apache Cordova – платформа розробки мобільних додатків, що дозволяє використовувати стандартні веб-технології, такі як *HTML5, CSS3* та *JavaScript* для крос-платформової розробки, уникаючи «рідної» мови програмування для кожної з мобільних платформ [7].

Основні переваги *Ionic*:

– розробка основної частини програмного забезпечення може вестись у звичайному браузері. Проте, для перевірки працездатності додатку відладку потрібно проводити безпосередньо в мобільному телефоні, або в емуляторі *Android*, або *iOS*;

– розробка додатку може вестись одночасно як для платформи *Android*, так і для *iOS*. Але при цьому є деякі обмеження, такі як особливості операційних систем що пов’язані зі стилями та плагінами;

– для розробки потрібні навички лише в *HTML, CSS, JavaScript* та фреймворку *Angular*(*Angular – JavaScript*-фреймворк з відкритим програмним кодом, призначений для розробки односторінкових додатків, що складаються з одної *HTML* сторінки з *JavaScript* і *CSS [8]*);

– для розробки інтерфейсу є багато готових компонентів що не потребують додаткових налаштувань: кнопки, поля для вводу, списки, перемикачі, таблиці тощо;

– доступна велика кількість плагінів, що дозволяє використовувати основні функції смартфону: камера, сканер відбитків пальців, геолокація, сповіщення тощо.

Основні недоліки технології «*Ionic*»:

– плагіни деяких бібліотек інколи можуть конфліктувати між собою

– відлагодження програми може бути доволі складним. Інколи складно визначити джерело помилка, оскільки сповіщення про несправність можуть бути неінформативними;

– готова програма може перестати працювати без видимої на те причини через те, що деякий компонент додатку в його папці виявляється пошкодженим [9].

Фреймворк *Ionic,* як і *Xamarin,* дозволяє створювати програми, чий інтерфейс має бути максимально наближеним до «рідного» на всіх мобільних платформах за рахунок вбудованих бібліотек.

1.3. Висновки до розділу

У даному розділі було розглянуто аналоги програмних засобів для мобільних та стаціонарних платформ для моніторингу розкладу руху залізничного транспорту. Було розглянуто засоби крос-платформової розробки додатків для мобільних операційних систем *Android* та *iOS*.

Для стаціонарних платформ було розглянуто сайти «*poezdato.net*» та офіційний сайт Укрзалізниці – «*uz.gov.ua*». Був зроблений короткий огляд даних сервісів та їх основних функцій.

Для мобільних платформ було розглянуто наступні програмні додатки: «Моя Електричка», «Розклади приміських поїздів України - *UZTrains*», «Яндекс.Электрички» та «Укрзалізниця». Був зроблений короткий огляд даних мобільних програмних засобів та їх основних функцій.

Було розглянуто наступні засоби крос-платформової розробки додатків для мобільних операційних систем *Android* та *iOS*: фреймворк *Qt,* фреймворк *Xamarin* та фреймворк *Ionic*. Були розглянути їх основні переваги та недоліки, їх принципи роботи. Було досліджено, у яких ситуаціях потрібно використовувати той чи інший засіб розробки для мобільних платформ: *Qt* підійде для розробки мобільних додатків, що потребують обробки великої кількості даних, *Xamarin* та *Ionic* підійде для розробки мобільних додатків, чий інтерфейс повинен бути максимально схожим на «рідний» для тої чи іншої мобільної оперативної системи.

РОЗДІЛ 2
ПРОЕКТУВАННЯ МОБІЛЬНОГО ДОДАТКУ МОНІТОРИНГУ РОЗКЛАДУ РУХУ ЗАЛІЗНИЧНОГО ТРАНСПОРТУ

2.1. Вибір платформи розробки та мови програмування

Для проектування програмного засобу було обрано мову програмування *C++* із фреймворком *Qt*, оскільки дана бібліотека має зручні інструменти для розробки додатку для мобільних операційних систем *Android* та *iOS*, а відладку коду та графічного інтерфейсу можна проводити на стаціонарних платформах. Розробка велася у середовищі розробки *QtCreator* із підключеним інструментарієм *AndroidSDK*. Графічний інтерфейс був написаний за допомогою мови *QML*.

Qt – крос-платформовий фреймворк для розробки програмного забезпечення для мови програмування *C++*. *Qt* складається із середовища розробки *QtCreator,* мови програмування *QML* для розробки графічного інтерфейсу, компіляторів *GCC, MinGW, Microsoft VC++* та *Clang*. Його основна перевага полягає в тому, що даний інструментарій дозволяє запускати написаний на ньому код на більшості сучасних операційних систем. Також він містить в собі всі основні класи, що можуть бути потрібними для розробки прикладного програмного забезпечення, починаючи з різноманітних контейнерів, закінчуючи елементами графічного інтерфейсу. Бібліотека дозволяє керувати потоками, працювати з мережею та надає розробнику можливість крос-платформового доступу до файлів [1].

C++ – мова програмування високого рівня, що підтримує три основні парадигми програмування: об’єктно-орієнтовану, процедурну та узагальнену.

Розроблена Б'ярном Страуструпом в *AT&T Bell Laboratories* у 1979 році. Вона базується на мові *C*. Основною причиною чому обираються саме *C++* у розробці програм – можливість ефективного управління оперативною пам’яттю пристрою, що допомагає знизити навантаження на апаратне забезпечення та пришвидшити час виконання певних алгоритмів [10].

Android SDK – універсальний засіб розробки мобільних додатків для операційної системи *Android*. Відмінною рисою від звичайних редакторів коду є наявність широких функціональних можливостей, що дозволяють запускати тестування і налагодження вихідних кодів, оцінювати роботу програми в режимі сумісності з різними версіями операційної системи *Android* і спостерігати результат в реальному часі. Підтримує велику кількість мобільних пристроїв, серед яких виділяють: мобільні телефони, планшетні комп'ютери тощо [11].

QtCreator – безкоштовне крос-платформове середовище розробки на мовах програмування *C++*, *C* та *QML*. Розроблена компанією *TrollTech (Digia)* для роботи з фреймворком *Qt*. Включає до себе графічний інтерфейс відладчика та візуальні засоби розробки інтерфейсу з використанням *QWidgets* або *QML*. Підтримує всі популярні компілятори такі як: *GCC*, *Clang*, *MinGW*, *MSVC*. Має свої засоби керування емуляторами різних пристроїв. Також у ньому присутня інтеграція з системами контролю версій: *GitHub*, *Subversion*, *Perforce* та *Mercurial* [12].

QML (*Qt Meta Language* або *Qt Modeling Language*) – декларативна мова програмування, заснована на *JavaScript* і призначена для розробки додатків, що роблять акцент на користувацький інтерфейс. Є частиною *Qt Quick*, середовища розробки користувацьких інтерфейсів, поширюваного разом з *Qt*. В основному використовується для створення додатків, орієнтованих на мобільні пристрої з сенсорним управлінням [13].

2.2. Проектування функцій мобільного додатку

Основні функції включають в себе операції, що користувач може виконувати під час використання додатку. При запуску програми повинні загрузитися всі дані з текстових файлів та створитися всі сутності, базуючись на вхідних даних. У додатку присутнє головне меню, що відкривається при натисканні на кнопку головного меню, або обранні пальцем зліва направо. У ньому є наступні пункти: пошук рейсів за номером; пошук рейсів за напрямком; пошук рейсів за станцією. Користувачеві пропонується обрати один із пунктів меню. При натисканні на нього, воно закривається та починає завантажуватися певна сторінка з певним набором функцій. Схема алгоритму роботи програмного засобу моніторингу розкладу руху залізничного транспорту зображена на рис. 2.1.

До основних функцій розроблюваної програми можна віднести: пошук рейсів за номером, пошук рейсів за напрямком та пошук рейсів, що проходять через обрану станцію.

При відкритті сторінки пошуку рейсів за номером користувачу представляється поле вводу, текстова підказка над ним та кнопка знизу. При вводі певних чисел програма у випадаючому списку пропонує вже існуючі рейси з їх напрямком. При натисканні на кнопку знизу з’являється розклад обраного потягу. Якщо номер рейсу не був знайдений, виводиться відповідна помилка.

При відкритті сторінки пошуку рейсів за напрямком користувачу представляється два поля вводу, дві текстові підказки над ними та кнопка знизу. При початку вводу у будь-яке поле програма пропонує у випадаючому списку вже існуючі станції. При натисканні на кнопку знизу з’являється розклад потягу, що проходить через обрані дві станції. Якщо хоча б одна станція не була знайдена, виводиться відповідна помилка.

При відкритті сторінки пошуку рейсів, що проходять через обрану станцію користувачу представляється поле вводу, текстова підказка над ним та кнопка знизу. При початку вводу у поле програма пропонує у випадаючому списку вже існуючі станції. При натисканні на кнопку знизу з’являється розклад потягу, що

проходить через обрану станцію. Якщо станція не була знайдена, виводиться відповідна помилка.

Рис. 2.1. Схема алгоритму роботи програмного засобу моніторингу розкладу руху залізничного транспорту

2.3. Проектування класів

При проектуванні даного додатку було розроблено наступні класи:

– клас *Routes*;

– клас *Stations*;

– клас *Wayroutes*;

– клас *ListModel*;

– клас *routesMenuModel*;

– клас *stationsMenuModel*.

Діаграма класів – статичне представлення структури моделі. Відображає статичні (декларативні) елементи, такі як: класи, типи даних, їх зміст та відношення. Діаграма класів служить для представлення статичної структури моделі системи в термінології програмування. На цій діаграмі показують класи, інтерфейси, об'єкти й кооперації, а також їхню взаємодію. Діаграма класів представлена на рис. 2.2.

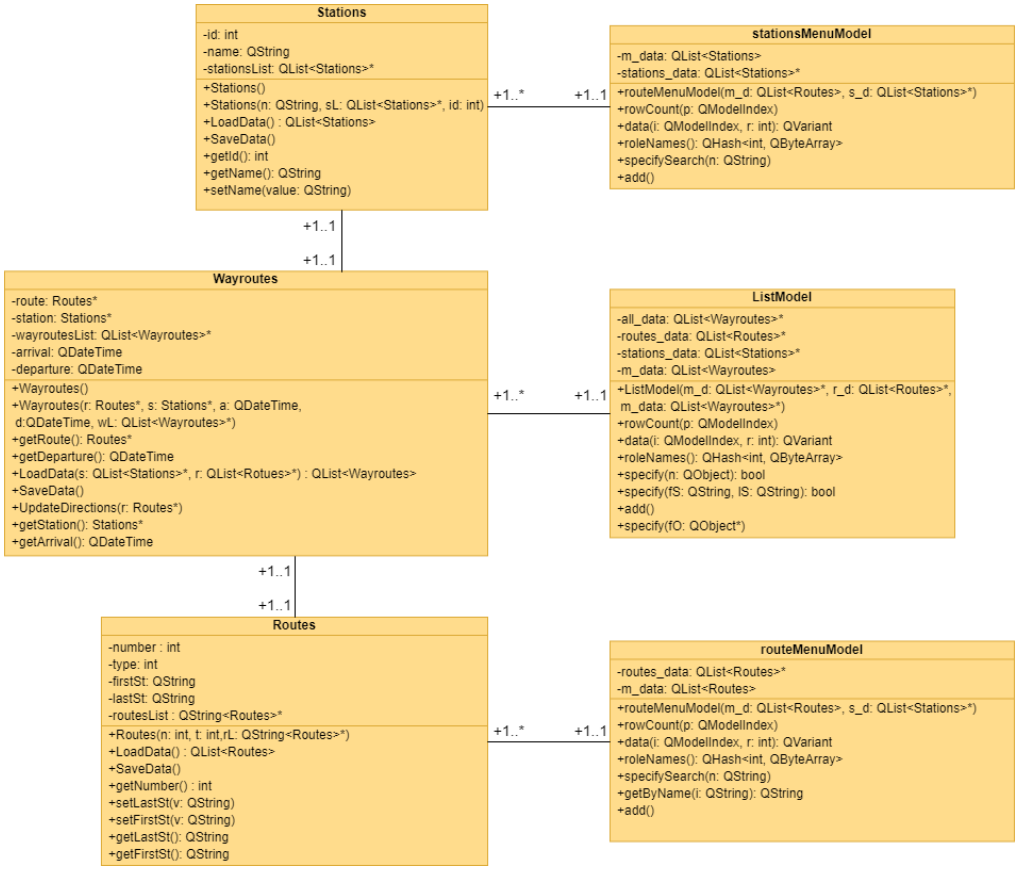

Рис. 2.2. Діаграма класів розроблюваного проекту

Клас *Routes* містить в собі змінні та методи, що призначенні для зберігання та обробки даних, пов’язаних із самими рейсами. Він має наступні змінні:

– номер рейсу - *number*;

– тип потягу - *type*;

– початкова станція рейсу - *firstSt*;

– кінцева станція рейсу - *lastSt*;

– покажчик на список усіх рейсів - *routesList*.

Номер потягу *number* має цілочисельний тип даних - *int*. Він використовується для визначення та пошуку маршруту слідкування певного рейсу.

Тип потягу *type* має цілочисельний тип даних - *int*. Він визначає тип потягу: міський або приміський.

Початкова станція рейсу *firstSt* має строковий тип даних - *QString*. Він визначає першу станцію, від якої потяг починає рух.

Кінцева станція рейсу *lastSt* має строковий тип даних - *QString*. Він визначає кінцеву станцію, де потяг закінчує свій рух.

Список усіх рейсів *routesList* – це покажчик на список, що містить в собі усі сутності типу *Routes* – *QList<Routes>**. Він призначений для алгоритмів у методах даного класу задля швидкого пошуку по усіх існуючих рейсів, що були ініціалізовані при запуску додатку.

Клас *Routes* має наступні методи:

– конструктор класу - *Routes*;

– завантаження даних - *LoadData*;

– отримання номеру рейсу - *getNumber*;

– встановлення початкової станції - *setFirstSt*;

– встановлення кінцевої станції - *setLastSt*;

– отримання початкової станції - *getFirstSt*;

– отримання кінцевої станції - *getLastSt*.

Конструктор класу *Routes* призначений для створення нової сутності типу *Routes*. Він приймає наступні значення: номер нового рейсу – *int _number*, тип рейсу – *int _type* та покажчик на список, що містить в собі усі рейси – *Qlist<Routes>* _routesList*. Даний метод не повертає жодних значень. При його виклику він встановлює значення змінних відповідно до переданих у метод даних

Метод завантаження даних *LoadData* призначений для зчитування усіх рейсів із відповідного файлу та створення списку нових сутностей типу *Routes* із отриманих даних. Він не приймає жодних значень та повертає список, що був створений при обробці відповідного файлу – *QList<Routes>*.

Метод отримання номеру рейсу *getNumber* виконує функцію отримання номеру потягу сутності, з якої був викликаний даний метод. Він не приймає жодних даних та повертає значення цілочисельного типу - *int*.

Метод встановлення початкової станції *setFirstSt* виконує функцію присвоювання початкової станції для сутності, з якої викликається даний метод. Він приймає значення строкового типу – *QString value* та нічого не повертає.

Метод встановлення кінцевої станції *setLastSt* виконує функцію присвоювання кінцевої станції для сутності, з якої викликається даний метод. Він приймає значення строкового типу – *QString value* та нічого не повертає.

Метод отримання початкової станції *getFirstSt* виконує функцію отримання початкової станції сутності, з якої був викликаний даний метод. Він не приймає жодних даних та повертає значення строкового типу - *QString*.

Метод отримання кінцевої станції *getLastSt* виконує функцію отримання кінцевої станції сутності, з якої був викликаний даний метод. Він не приймає жодних даних та повертає значення строкового типу - *QString*.

Клас *Stations* містить в собі змінні та методи, що призначенні для зберігання та обробки даних, пов’язаних із станціями. Він має наступні змінні:

– порядковий номер станції - *id*;

– назва станції - *name*;

– покажчик на список усіх станцій - *stationsList*.

Порядковий номер станції *id* має цілочисельний тип даних - *int*. Він використовується під час завантаження даних із файлів для зв'язування сутностей класу *Wayroutes* із об'єктами типу *Stations*.

Назва станції *name* має строковий тип даних – *QString*. Він призначений для зберігання назви певної станції.

Список усіх станцій *stationsList* – це покажчик на список, що містить в собі усі сутності типу *Stations*. Він призначений для алгоритмів у методах даного класу задля швидкого пошуку по усіх існуючих станціях, що були ініціалізовані при запуску додатку.

Клас *Stations* має наступні методи:

– конструктор класу - *Stations*;

– завантаження даних - *LoadData*;

– отримання порядкового номеру станції - *getId*;

– отримання назви станції - *getName*;

– встановлення назви станції - *setName*.

Конструктор класу *Stations* призначений для створення нової сутності типу *Stations*. Він приймає наступні значення: назва нової станції – *QString _name*, покажчик на список, що містить в собі усі станції – *Qlist<Stations>* _stationsList* та порядковий номер рейсу – *int _id*. Даний метод не повертає жодних значень. При його виклику він встановлює значення змінних відповідно до переданих у метод даних

Метод завантаження даних *LoadData* призначений для зчитування усіх станцій із відповідного файлу та створення списку нових сутностей типу *Stations* із отриманих даних. Він не приймає жодних значень та повертає список, що був створений при обробці відповідного файлу – *QList<Stations>*.

Метод отримання порядкового номеру станції *getId* виконує функцію отримання номеру сутності, з якої був викликаний даний метод. Він не приймає жодних даних та повертає значення цілочисельного типу - *int*.

Метод отримання назви станції *getName* виконує функцію отримання назви сутності, з якої був викликаний даний метод. Він не приймає жодних даних та повертає значення строкового типу - *QString*.

Метод встановлення назви станції *setName* виконує функцію присвоювання станції для сутності, з якої викликається даний метод. Він приймає значення строкового типу *QString value* та нічого не повертає.

Клас *Wayroutes* містить в собі змінні та методи, що призначенні для зберігання та обробки даних, пов'язаних із зупинками, що здійснює кожен із рейсів. Він має наступні змінні:

– покажчик на об'єкт рейсу - *route*;

– покажчик на об'єкт станції - *station*;

– дата та час прибуття потягу- *arrival*;

– дата та час відправлення потягу - *departure*;

– покажчик на список усіх зупинок - *wayroutesList*.

Тип даних покажчика на об'єкт рейсу *route* – *Routes**. Він вказує на сутність рейсу для даної зупинки.

Тип даних покажчика на об'єкт станції *station* – *Stations**. Він вказує на сутність зупинки для даної зупинки.

Змінна дати та часу прибуття потягу на певну станцію *arrival* – *QDateTime.* Вона зберігає в собі дату та час прибуття потягу на станцію, що зв'язана з даною зупинкою потягу.

Змінна дати та часу прибуття відправлення від певної станції *departure* – *QDateTime*. Вона зберігає в собі дату та час відправлення потягу від станції, що зв'язана з даною зупинкою потягу.

Список усіх зупинок *wayroutesList* – це покажчик на список, що містить в собі усі сутності типу *Wayroutes* – *QList<Wayroutes>**. Він призначений для алгоритмів у методах даного класу задля швидкого пошуку по усіх існуючих зупинках, що були ініціалізовані при запуску додатку.

Клас *Wayroutes* має наступні методи:

– конструктор класу - *Wayroutes*;

– завантаження даних - *LoadData*;

– отримання сутності рейсу - *getRoute*;

– отримання сутності станції- *getStation*;

– отримання дати та часу прибуття - *getArrival*;

– отримання дати та часу відправлення - *getDeparture*;

– оновлення початкової та кінцевої станції рейсу - *UpdateDirections*;

Конструктор класу *Wayroutes* призначений для створення нової сутності типу *Wayroutes*. Він приймає наступні значення: покажчик на сутність рейсу – *Routes* _route*, покажчик на сутність станції – *Stations* station*, дата та час прибуття на станцію – *QDateTime _arrival*, дата та час відправлення від станції – *QDateTime _departure* та покажчик на список, що містить в собі усі зупинки – *QList<Wayroutes>* _wayroutesList*. Даний метод не повертає жодних значень. При його виклику він встановлює значення змінних відповідно до переданих в аргументи методу даних.

Метод завантаження даних *LoadData* призначений для зчитування усіх рейсів із відповідного файлу та створення списку нових сутностей типу *Wayroutes* із отриманих даних. Він не приймає жодних значень та повертає список, що був створений при обробці відповідного файлу – *QList<Wayroutes>*.

Метод отримання сутності рейсу *getRoute* виконує функцію отримання сутності рейсу для даної зупинки. Він не приймає жодних даних та повертає покажчик об'єкту рейсу – *Routes**.

Метод отримання сутності станції *getStation* виконує функцію отримання сутності станції для даної зупинки. Він не приймає жодних даних та повертає покажчик об'єкту станції – *Stations**.

Метод отримання дати та часу прибуття *getArrival* виконує функцію отримання дати та часу прибуття на дану станцію. Він не приймає жодних даних та повертає значення типу *QDateTime*.

Метод отримання дати та часу прибуття *getDeparture* виконує функцію отримання дати та часу відправлення від даної станції. Він не приймає жодних даних та повертає значення типу *QDateTime*.

Метод оновлення початкової та кінцевої станції рейсу *UpdateDirections* виконує функцію присвоювання зв'язаному з даною зупинкою рейсу нових значень змінних початкової станції *firstSt* та кінцевої станції *lastSt*. Даний метод

викликається у конструкторі *Wayroutes*. Він приймає покажчик рейсу для якого потрібне оновлення – *Routes* dataRoute* та нічого не повертає.

Клас *ListModel* містить в собі змінні та методи, що призначенні для специфікації та виводу розкладу потягів на екран користувача. Він має наступні змінні:

– покажчик на список об'єктів зупинок - *all_data*;

– покажчик на список об'єктів рейсів - *routes_data*;

– покажчик на список об'єктів станцій - *stations_data*;

– список об'єктів зупинок, що виводяться користувачеві - *m_data.*

Тип даних покажчика на список об'єктів зупинок *all_data* – *QList<Wayroutes>**. Він вказує на список сутностей усіх зупинок.

Тип даних покажчика на список об'єктів зупинок routes*_data* – *QList<Routes>**. Він вказує на список сутностей усіх рейсів.

Тип даних покажчика на список об'єктів зупинок *stations_data* – *QList<Stations>**. Він вказує на список сутностей усіх станцій.

Тип даних списку об'єктів зупинок, що виводяться користувачеві *m_data* – *QList<Wayroutes>*. У ньому зберігаються усі зупинки, що потрібно виводити користувачу згідно критеріїв, що він обрав.

Клас *ListModel* має наступні методи:

– конструктор класу - *ListModel*;

– підрахунок кількості записів у списку - *rowCount*;

– формування строк що виводяться користувачу - *data*;

– отримання назви ролей - *roleNames*;

– специфікація видачі розкладу за критеріями користувача - *specify*;

– оновлення даних у користувацькому інтерфейсі - *add*;

Конструктор класу *ListModel* призначений для створення нової сутності типу *ListModel*. Він приймає наступні значення: покажчик на список, що містить усі зупинки – *QList<Wayroutes>* _m_data*; покажчик на список, що містить усі рейси – *QList<Routes>* _routes_data;* покажчик на список, що містить усі станції – *QList<Stations>* _stations_data*. Даний метод не повертає жодних

значень. При його виклику він встановлює значення змінних відповідно до переданих у метод даних

Метод підрахунку кількості записів у списку *rowCount* є автоматично згенерованим та призначений для отримання кількості строк у списку. Він не приймає жодних значень та повертає значення цілочисельного типу – *int*.

Метод формування строк, що виводяться користувачу *data* є автоматично згенерованим та виконує функцію виводу певної інформації, що відповідає заданої ролі. Він приймає наступні значення: індекс потрібної строки у списку – *QModelIndex index*, роль даних, що потрібно вивести користувачу – *int role*, не повертає жодних значень.

Метод отримання назви ролей *roleNames* виконує функцію отримання всіх ролей, що потрібні для розподілу даних для виводу. Він не приймає жодних даних та повертає контейнер з ролями – *QHash<int, QByteArray>*.

Метод специфікації видачі розкладу за критеріями користувача *specify* є перевантаженою функцією, тому є три її реалізації. Перша реалізація записує до списку зупинок *m_data* усі зупинки обраного рейсу. Вона приймає змінну номеру потрібного рейсу строкового типу – *QString rNumber*. Друга реалізація відповідає за записування до списку зупинок *m_data* маршруту, що відправляється від однієї станції та прибуває на іншу. Вона приймає наступні дані: змінна назви першої станції строкового типу – *QString _firstStation*, змінна назви другої станції строкового типу – *QString _lastStation*. Третя реалізація записує до списку зупинок *m_data* маршрут рейсу, що проходить через певну станцію. Вона приймає покажчик на об’єкт інтерфейсу, у який користувач вводить свій запит – *QObject* _firstStationObject*. Усі реалізації повертають змінну типу *bool*. Якщо специфікація була успішна – повертається *true*, інакше – повертається *false*.

Метод оновлення даних у користувацькому інтерфейсі *add* є автоматично згенерованим. Він викликається при специфікації даних у списку зупинок *m_data*. Дана функція не приймає та не повертає жодних даних.

Клас *routeMenuModel* містить в собі змінні та методи, що призначенні для виводу існуючих рейсів у випадаючому списку при вводі користувачем рейсів. Він має наступні змінні:

– покажчик на список об’єктів рейсів – *routes_data*;

– список об’єктів рейсів, що виводяться користувачеві – *m_data*.

Тип даних покажчика на список об’єктів рейсів *routes_data* – *QList<Routes>**. Він вказує на список сутностей усіх рейсів.

Тип даних списку об’єктів рейсів, що виводяться користувачеві *m_data* – *QList<Routes>*. У ньому зберігаються усі рейси, що потрібно виводити користувачу у випадаючому списку при вводі в текстове поле.

Клас *routeMenuModel* має наступні методи:

– конструктор класу - *routeMenuModel*;

– підрахунок кількості записів у списку - *rowCount*;

– формування строк, що виводяться користувачу- *data*;

– отримання назви ролей - *roleNames*;

– специфікація видачі номеру рейсу – *specifySearch*;

– отримання номеру рейсу із строки – *getByName*;

– формування строки напрямку потягу – *getRouteStations*;

– оновлення даних у користувацькому інтерфейсі – *add*;

Конструктор класу *routeMenuModel* призначений для створення нової сутності типу *routeMenuModel*. Він приймає значення покажчика на список, що містить усі рейси – *QList<Routes>* _m_data*. Даний метод не повертає жодних значень. При його виклику він встановлює значення змінних відповідно до переданих у метод даних

Метод підрахунку кількості записів у списку *rowCount* є автоматично згенерованим та призначений для отримання кількості строк у списку. Він не приймає жодних значень та повертає значення цілочисельного типу – *int*.

Метод формування строк що виводяться користувачу *data* є автоматично згенерованим та виконує функцію виводу певної інформації, що відповідає заданої ролі. Він приймає наступні значення: індекс потрібної строки у списку –

QModelIndex index, роль даних що потрібно вивести користувачу – *int role*, не повертає жодних значень.

Метод отримання назви ролей *roleNames* виконує функцію отримання всіх ролей, що потрібні для розподілу даних для виводу. Він не приймає жодних даних та повертає контейнер з ролями – *QHash<int, QByteArray>*.

Метод специфікації видачі номеру рейсу *specifySearch* призначений для специфікації рейсів за даними, що користувач вводить у поле вводу та видачі їх у випадаючому списку. Він приймає змінну номеру рейсу строкового типу – *QString rNumber*. Функція повертає змінну типу *bool*. Якщо специфікація була успішна – повертається *true*, інакше – повертається *false*.

Метод отримання номеру рейсу із строки *getByName* призначений для отримання рейсу із загальної строки, що складається з його номеру та напрямку. Він приймає змінну строкового типу *QString _index* та повертає змінну строкового типу *QString*.

Метод формування строки напрямку потягу *getRouteStations* призначений для отримання напрямку певного рейсу, що складається з початкової та кінцевої станцій. Він приймає змінну номеру рейсу строкового типу – *QString _index* та повертає змінну строкового типу *QString*.

Метод оновлення даних у користувацькому інтерфейсі *add* є автоматично згенерованим. Він викликається при специфікації даних у списку зупинок *m_data*. Дана функція не приймає та не повертає жодних даних.

Клас *stationMenuModel* містить в собі змінні та методи, що призначенні для виводу існуючих станцій у випадаючому списку при вводі користувачем станцій. Він має наступні змінні:

– покажчик на список об’єктів станцій – *stations_data*;

– список об’єктів станцій, що виводяться користувачеві – *m_data*.

Тип даних покажчика на список об’єктів станцій stations*_data* – *QList<Stations>**. Він вказує на список сутностей усіх станцій.

Тип даних списку об’єктів станцій, що виводяться користувачеві *m_data* – *QList<Stations>*. У ньому зберігаються усі станції, що потрібно виводити користувачу у випадаючому списку при вводі в текстове поле.

Клас *stationMenuModel* має наступні методи:

– конструктор класу - *stationMenuModel*;

– підрахунок кількості записів у списку - *rowCount*;

– формування строк що виводяться користувачу - *data*;

– отримання назви ролей - *roleNames*;

– специфікація видачі назви станції - *specifySearch*;

– оновлення даних у користувацькому інтерфейсі - *add*.

Конструктор класу *stationMenuModel* призначений для створення нової сутності типу *stationMenuModel*. Він приймає значення покажчика на список, що містить усі станції – *QList<Stations>* _m_data*. Даний метод не повертає жодних значень. При його виклику він встановлює значення змінних відповідно до переданих у метод даних.

Метод підрахунку кількості записів у списку *rowCount* є автоматично згенерованим та призначений для отримання кількості строк у списку. Він не приймає жодних значень та повертає значення цілочисельного типу – *int*.

Метод формування строк що виводяться користувачу *data* є автоматично згенерованим та виконує функцію виводу певної інформації, що відповідає заданої ролі. Він приймає наступні значення: індекс потрібної строки у списку – *QModelIndex index*, роль даних що потрібно вивести користувачу – *int role*, не повертає жодних значень.

Метод отримання назви ролей *roleNames* виконує функцію отримання всіх ролей, що потрібні для розподілу даних для виводу. Він не приймає жодних даних та повертає контейнер з ролями – *QHash<int, QByteArray>*.

Метод специфікації видачі назв станцій *specifySearch* призначений для специфікації станцій за даними, що користувач вводить у поле вводу та видачі їх у випадаючому списку. Він приймає змінну назви станції строкового типу – *QString stationName*. Функція повертає змінну типу *bool*. Якщо специфікація була успішна – повертається *true*, інакше – повертається *false*.

Метод оновлення даних у користувацькому інтерфейсі *add* є автоматично згенерованим. Він викликається при специфікації даних у списку зупинок *m_data*. Дана функція не приймає та не повертає жодних даних.

2.4. Висновки до розділу

У даному розділі було спроектовано мобільний додаток для моніторингу розкладу руху залізничного транспорту. Було обрано мову програмування та платформу розробки, визначено основні функції програмного засобу та спроектовано основні класи програмного засобу.

Було обрано мову програмування та платформу розробки: *C++* із фреймворком *Qt,* мову програмування для розробки графічного інтерфейсу *QML,* середовище розробки *QtCreator* із засобом розробки мобільних додатків *AndroidSDK*.

Основними функціями програмного засобу було визначено: пошук рейсу за його номером, пошук рейсів за напрямком та пошук рейсів за станцією. Для кожної функції було описано їх алгоритм роботи та їхню взаємодію з користувачем.

Було спроектовано наступні класи: *Routes, Stations, Wayroutes, ListModel*, *routeMenuModel* та *stationMenuModel*. Були визначенні їх змінні та методи. Було описано, які типи та назви мають змінні класів, які аргументи приймає кожна функція та які значення вона повертає.

РОЗДІЛ 3
РОЗРОБКА МОБІЛЬНОГО ДОДАТКУ МОНІТОРИНГУ РОЗКЛАДУ РУХУ ПРИМІСЬКОГО ЗАЛІЗНИЧНОГО ТРАНСПОРТУ

3.1. Склад файлів проекту

Програмний засіб було створено у середовищі розробки *Qt Creator*. Для створення проекту потрібно відкрити «Файл» та обрати «Создать файл или проект». Був обраний шаблон проекту «Приложение *Qt Quick* - пустое». Після цього було введено назву проекту – *train_schedule* та обрано папку *C:\projectTrainSchedule*. Систему зборки було обрано *qmake*. Далі обрано мінімально необхідну версію *Qt – Qt 5.12*. Після цього не було обрано файл перекладу додатку. У вікні комплектів обрано *Android Qt 5.12 Clang Multi-Abi*. На наступному етапі контроль версій обрано не було. У результаті цього, було створено проект у папці *C:\projectTrainSchedule\train_schedule.*

Папка *train_schedule* містить автоматично створені файли: *train_schedule.pro, qml.qrc, main.qml* та *main.cpp.* У ході розробки проекту були створені наступні файли: *listmodel.cpp, listmodel.h, routemenumodel.cpp, routemenumodel.h, routes.cpp, routes.h, stationmenumodel.cpp, stationmenumodel.h, stations.cpp, stations.h, wayroutes.cpp, wayroutes.h, listRouteDisplay.qml, timetableDisplay.qml, stationDisplay.qml, welcomeScreen.qml, Routes.txt, Stations.txt, Wayroutes.txt.*

Файл *train_schedule.pro* містить в собі налаштування проекту. У ньому вказуються наступні налаштування: файли, що входять до проекту; використовувані бібліотеки *Qt;* версія мови програмування *C++*; налаштування компілятора.

qml.qrc – файл ресурсів проекту. Він містить в собі всі файли *qml*, текстові та мультимедійні файли. При зборці проекту *qml.qrc* запаковується в програму.

Файл *main.qml* містить в собі код основного інтерфейсу мобільного додатку, а саме: верхню панель, що містить в собі назву поточної сторінки та кнопку меню; висувне меню; елемент *Loader*, що призначений для завантаження різних сторінок інтерфейсу програми.

Файл *main.cpp* містить в собі код, що виконується при запуску програми. У ньому викликаються функції налаштування програми та оголошуються сутності, що використовуються протягом користування додатком. Наприклад, ініціалізація списків об'єктів класів *Routes, Stations* та *Wayroutes*; ініціалізація *ListModel, routeMenuModel* та *stationMenuModel* та зв'язування їх із *QML* файлами.

Файл *listmodel.cpp* містить реалізацію тіла класу *ListModel*. У ньому міститься реалізація методів, конструктор класу та функції для отримання та встановлення деяких змінних.

Файл *listmodel.h* містить визначення класу *ListModel*. Тобто він визначає його змінні: *QList<Wayroutes> *all_data, QList<Routes> *routes_data QList<Stations> *stations_data* та *QList<Wayroutes> m_data*. Також він містить прототипи функцій даного класу: *listModel(QList<Wayroutes> *_m_data, QList<Routes> *_routes_data, QList<Stations> *_stations_data, QObject *parent = 0); int rowCount(const QModelIndex &parent) const; QVariant data(const QModelIndex &index, int role) const; QHash<int, QByteArray> roleNames() const; Q_INVOKABLE bool specify(QString rNumber); Q_INVOKABLE bool specify(QString _firstStation, QString _lastStation); Q_INVOKABLE void add().*

Файл *routemenumodel.cpp* містить реалізацію тіла класу *routeMenuModel*. У ньому міститься реалізація методів, конструктор класу та функції для отримання та встановлення деяких змінних.

Файл *routemenumodel.h* містить визначення класу *routeMenuModel*. Тобто він визначає його змінні: *QList<Routes> *routes_data* та *QList<Routes> m_data*. Також він містить прототипи функцій даного класу: *routeMenuModel(QList<Routes> *_m_data, QObject *parent = 0); int rowCount*

(const QModelIndex &parent) const; QVariant data(const QModelIndex &index, int role) const; QHash<int, QByteArray> roleNames() const; Q_INVOKABLE bool specifySearch(QString rNumber); Q_INVOKABLE QString getByName(QString _index); Q_INVOKABLE QString getRouteStations(QString _index); Q_INVOKABLE void add().

Файл *routes.cpp* містить реалізацію тіла класу *Routes*. У ньому міститься реалізація методів, конструктор класу та функції для отримання та встановлення деяких змінних.

Файл *routes.h* містить визначення класу *Routes*. Тобто він визначає його змінні: *int number, int type, QString firstSt, QString lastSt* та *QList<Routes>* routesList*. Також він містить прототипи функцій даного класу: *Routes(int _number, int _type, QList<Routes>* _routesList); QList<Routes> LoadData(); int getNumber() const; void setLastSt(const QString &value); void setFirstSt(const QString &value); QString getFirstSt() const; QString getLastSt() const.*

Файл *stationmenumodel.cpp* містить реалізацію тіла класу *stationMenuModel*. У ньому міститься реалізація методів, конструктор класу та функції для отримання та встановлення деяких змінних.

Файл *stationmenumodel.h* містить визначення класу *stationMenuModel*. Тобто він визначає його змінні: *QList<Stations> *stations _data* та *QList<Stations> m_data*. Також він містить прототипи функцій даного класу: *stationMenuModel(QList<Stations> *_m_data, QObject *parent = 0); int rowCount(const QModelIndex &parent) const; QVariant data(const QModelIndex &index, int role) const; QHash<int, QByteArray> roleNames() const; Q_INVOKABLE bool specifySearch(QString stationName); Q_INVOKABLE void add().*

Файл s*tations.cpp* містить реалізацію тіла класу *Stations*. У ньому міститься реалізація методів, конструктор класу та функції для отримання та встановлення деяких змінних.

Файл *stations.h* містить визначення класу *Stations*. Тобто він визначає його змінні: *int id, QString name* та *QList<Stations>* stationsList.* Також він містить прототипи функцій даного класу: *Stations(QString _name, QList<Stations>**

_stationsList, int _id = 0); QList<Stations> LoadData(); int getId() const; QString getName() const; void setName(const QString &value).

Файл *wayroutes.cpp* містить реалізацію тіла класу *Wayroutes*. У ньому міститься реалізація методів, конструктор класу та функції для отримання та встановлення деяких змінних.

Файл *wayroutes.h* містить визначення класу *Wayroutes*. Тобто він визначає його змінні*: Routes* route, Stations* station, QList<Wayroutes>* wayroutesList, QDateTime arrival, QDateTime departure*. Також він містить прототипи функцій даного класу: *Wayroutes(Routes* _route, Stations* _station, QDateTime _arrival, QDateTime _departure, QList<Wayroutes>* _wayroutesList); Routes *getRoute() const; QDateTime getDeparture() const; QList<Wayroutes> LoadData(QList<Stations>* dataStations, QList<Routes>* dataRoutes); void UpdateDirections(Routes* dataRoute); Stations *getStation() const; QDateTime getArrival() const.*

Файл *listRouteDisplay.qml* містить код, написаний на мові програмування *QML* та *JavaScript*. Він відповідає за розташування елементів у графічному інтерфейсі на сторінці пошуку потягу за номером рейсу.

Файл *timetableDisplay.qml* містить код, написаний на мові програмування *QML* та *JavaScript*. Він відповідає за розташування елементів у графічному інтерфейсі на сторінці пошуку потягів, що прямують від однієї станції до іншої.

Файл *stationDisplay.qml* містить код, написаний на мові програмування *QML* та *JavaScript*. Він відповідає за розташування елементів у графічному інтерфейсі на сторінці пошуку потягів, що проходять через станцію.

Файл *welcomeScreen.qml* містить код, написаний на мові програмування *QML*. Він відповідає за розташування елементів у графічному інтерфейсі на самій першій сторінці додатку, що виводить інструкцію як почати роботу з програмою.

Файл *Routes.txt* містить дані, що потрібно завантажити в сутності класу *Routes* при запуску програми. Він має наступну структуру даних: номер рейсу, тип рейсу.

Файл *Stations.txt* містить дані, що потрібно завантажити в сутності класу *Stations* при запуску програми. Він має наступну структуру даних: назва станції, *id* станції.

Файл *Wayroutes.txt* містить дані, що потрібно завантажити в сутності класу *Wayroutes* при запуску програми. Він має наступну структуру даних: *id* станції, номер рейсу, дата та час прибуття потягу, дата та час відправлення потягу.

3.2. Розробка інтерфейсу мобільного додатку

Основний інтерфейс користувача мобільного додатку моніторингу розкладу приміського залізничного транспорту реалізовано у файлі *main.qml*. Використані елементи графічного інтерфейсу були взяті з бібліотек *QtQuick* та *QtContorol*. Під час запуску програми елементи відображаються на екрані мобільного пристрою, а саме: верхня панель додатку з назвою сторінки та кнопкою меню, меню, що реалізовано за допомогою елементу *Drawer* та елемент *Loader*. *Drawer* – висувне меню, що можна відкрити при обранні зліва-направо. *Loader* – елемент, що завантажує різні сторінки додатку з файлів типу *qml*. Верхня панель додатку показана на рис. 3.1. При запуску програмного засобу елемент *Loader* відображає сторінку з інструкцією як почати роботу з додатком, що міститься в файлі *welcomeScreen.qml*.

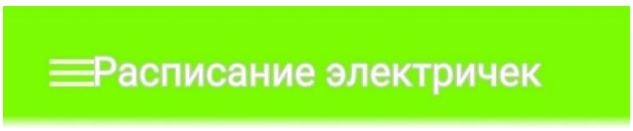

Рис. 3.1. Верхня панель додатку

Меню додатку реалізовано в елементі *Drawer*. Елементи його меню складаються з наступних елементів: *Rectangle* та *Label*. *Rectangle* – елемент, що має прямокутну форму та може виступати як об’єднуючим елементом для інших елементів, або як елемент інтерфейсу користувача. *Label* – елемент інтерфейсу, що призначений для виводу тексту на екрані користувача. У ньому є наступні пункти меню: пошук за номером рейсу, пошук за напрямком та пошук по станції. При обранні пункту пошуку за номером рейсу елемент *Loader*

завантажує сторінку, що описана в файлі *listRouteDisplay.qml*. При обранні пункту пошук за напрямком елемент *Loader* завантажує сторінку, що описана в файлі *timetableDisplay.qml*. При обранні пункту пошуку за станцією елемент *Loader* завантажує сторінку, що описана в файлі *stationsDisplay.qml*. Вигляд меню додатку представлений на рис. 3.2.

Рис. 3.2. Меню додатку

Інтерфейс початкової сторінки додатку описаний у файлі *welcomeScreen.qml*. Інтерфейс складається з двох елементів *Label*. Користувачеві виводиться текст, що потрібно відкрити меню та обрати йому потрібний пункт. Вигляд сторінки з інструкцією представлений на рис. 3.3.

Рис. 3.3. Вікно сторінки з інструкцією

Інтерфейс сторінки пошуку рейсу за його номером описаний у файлі *listRouteDisplay.qml*. Інтерфейс складається з наступних елементів:, *Label* з назвою *routeInputLabel*, *TextInput, ListView* та *Component*. *InputText* – елемент, що надає користувачеві можливість вводити текст, тобто він використовується для отримання ввідних даних від користувача. *ListView* – це елемент, що формує список, що отримується зі змінної класу *ListModel* – *m_data*. *Component* – це елемент, що визначає, як виглядає елемент у списку. Вигляд сторінки пошуку рейсу за його номером представлений на рис. 3.4.

Інтерфейс сторінки пошуку рейсів за напрямком описаний у файлі *timetableDisplay.qml*. Він складається з наступних елементів, *Rectangle*, *Label*, *InputText, ListView* та *Component.* Вигляд сторінки пошуку рейсів за напрямком представлений на рис. 3.5.

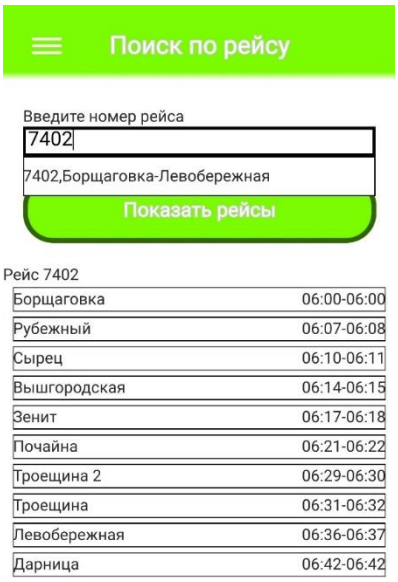

Рис. 3.4. Вікно сторінки пошуку рейсу за його номером

Інтерфейс сторінки пошуку рейсів за станцією описаний у файлі *stationDisplay.qml.* Він складається з наступних елементів, *Rectangle, Label*, *InputText, ListView* та *Component.* Вигляд сторінки пошуку рейсів за станцією представлений на рис. 3.6.

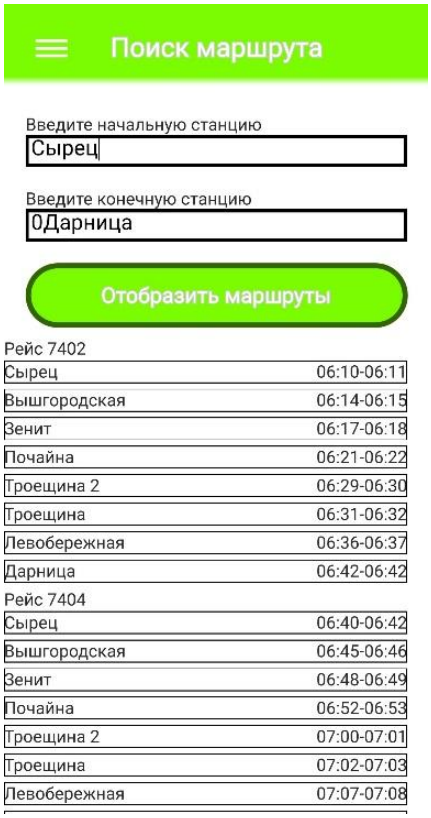

Рис. 3.5. Вікно сторінки пошуку рейсів за напрямком

Рис. 3.6. Вікно сторінки пошуку рейсів за станцією

3.3. Розробка модулів мобільного додатку

3.3.1. Модуль ініціалізації змінних та завантаження файлів

Модуль ініціалізації змінних та завантаження файлів розроблено для створення усіх потрібних об’єктів, контейнерів сутностей, завантаження даних із файлів до цих об’єктів, зв’язування сутностей з графічним інтерфейсом. Частина цього модуля реалізована у файлі *main.cpp*, а саме: відбувається створення трьох контейнерів з об’єктами класів *Routes, Stations* та *Wayroutes*; відбувається створення сутностей класів *Routes, Stations* та *Wayroutes,* що використовуються для завантаження даних у раніше створених контейнерах; відбувається створення об’єктів класів *ListModel, routeMenuModel* та *stationMenuModel* для зв’язування їх з файлами графічного інтерфейсу; відбувається виклик автоматично створених функцій для налаштування роботи додатку. Інша частина реалізована в класах *Routes, Stations, Wayroutes, ListModel, routeMenuModel* та *stationMenuModel.*

У методі *LoadData* класу *Routes* реалізований алгоритм завантаження даних з файлу *Routes.txt*. Дана функція не приймає жодних значень та повертає список із сутностями типу *Routes* – *QList<Routes>*. Спочатку створюється новий контейнер типу *QList<Routes>* з назвою *foo*, до якого будуть записані всі далі завантажені з файлу рейси. Потім програма відкриває файл *Routes.txt* та за допомогою циклу починає зчитувати усі строки, що містяться в файлі у вигляді строкового типу *QString*. Так як кожен запис зберігається у форматі “номер рейсу, тип рейсу”, програма розділяє цей запис на дві різні строки та передає їх у конструктор класу *Routes* у вигляді двох аргументів, попередньо перевівши їх у цілочисельний тип *int*. Після цього, створюється нова сутність типу *Routes*, що додається до списку *QList<Routes> foo*. В кінці алгоритму метод *LoadData* повертає контейнер *QList<Routes> foo*.

У конструкторі класу *Routes* реалізовано функцію створення нової сутності типу *Routes*. Дана функція приймає наступні значення: змінну номеру рейсу *number* цілочисельного типу – *int*; змінну типу рейсу *type* цілочисельного

типу – *int*; покажчик на список *routesList*, що містить усі сутності типу *Routes* – *QList<Routes>**. Функція не повертає жодних значень. При її виклику створюється нова сутність типу *Routes*, змінні якої присвоюються згідно отриманих аргументів, також створений об'єкт додається до списку *routesList*.

У конструкторі класу *Stations* реалізовано функцію створення нової сутності типу *Stations*. Дана функція приймає наступні значення: змінну назви станції *name* строкового типу - *QString*; покажчик на список *stationsList*, що містить усі сутності типу *Stations* – *QList<Stations>*;* змінну порядкового номеру станції *id* цілочисельного типу – *int*. Функція не повертає жодних значень. При її виклику створюється нова сутність типу *Stations*, змінні якої присвоюються згідно отриманих аргументів, також створений об'єкт додається до списку *stationsList*.

У методі *LoadData* класу *Stations* реалізований алгоритм завантаження даних з файлу *Stations.txt*. Дана функція не приймає жодних значень та повертає список із сутностями типу *Stations* – *QList<Stations>*. Спочатку створюється новий контейнер типу *QList<Stations>* з назвою *foo*, до якого будуть записані всі далі завантажені з файлу станції. Потім програма відкриває файл *Stations.txt* та за допомогою циклу починає зчитувати усі строки, що містяться в файлі у вигляді строкового типу *QString*. Так як кожен запис зберігається у форматі "назва станції, *id* станції", програма розділяє цей запис на дві різні строки та передає їх у конструктор класу *Routes* у вигляді двох аргументів, попередньо перевівши *id* станції у цілочисельний тип *int*. Після цього, створюється нова сутність типу *Stations*, що додається до списку *QList<Stations> foo*. В кінці алгоритму метод *LoadData* повертає контейнер *QList<Stations> foo*.

У конструкторі класу *Wayroutes* реалізовано функцію створення нової сутності типу *Wayroutes*. Дана функція приймає наступні значення: покажчик на список *_route*, що містить усі сутності типу *Routes – QList<Routes>*;* покажчик на список *_station*, що містить усі сутності типу *Stations – QList<Stations>*;* змінну *_arrival* типу *QDateTime*, що містить дату та час прибуття потягу на станцію; змінну *_departure* типу *QDateTime*, що містить дату та час відправлення потягу від станції; покажчик на список *_wayroutesList*, що містить усі сутності

типу *Wayroutes – QList<Wayroutes>**. Функція не повертає жодних значень. При її виклику створюється нова сутність типу *Stations*, змінні якої присвоюються згідно отриманих аргументів, викликається функція *UpdateDirections* для оновлення напрямку для рейсу, що зв'язаний з даною зупинкою, а також створений об'єкт додається до списку *stationsList*.

У методі *LoadData* класу *Wayroutes* реалізований алгоритм завантаження даних з файлу *Wayroutes.txt*. Дана функція приймає наступні значення: покажчик на список *dataStations* типу *QList<Stations>**, що містить усі станції; покажчик на список *dataRoutes* типу *QList<Routes>**, що містить усі рейси. Вона повертає список із сутностями типу *Wayroutes – QList<Wayroutes>*. Спочатку створюється новий контейнер типу *QList<Wayroutes>* з назвою *foo,* до якого будуть записані всі далі завантажені з файлу, що містить усі зупинки. Потім програма відкриває файл *Wayroutes.txt* та за допомогою циклу починає зчитувати усі строки, що містяться в файлі у вигляді строкового типу *QString*. Так як кожен запис зберігається у форматі " *id* станції, номер рейсу, дата та час прибуття потягу, дата та час відправлення потягу", програма розділяє цей запис на чотири різні строки. Після цього програма веде пошук потрібного рейсу згідно отриманого номеру рейсу з файлу у списку *dataRoutes* та передає її у покажчик *tempRoute* типу *Routes**.Після цього програма веде пошук потрібної станції згідно отриманого *id* станції з файлу у списку *dataStations* та передає її у покажчик *tempStation* типу *Stations**. Потім програма переводить отримані дати та час прибуття і відправлення з типу *QString* у тип *QDateTime*. Після цього програма передає усі чотири змінні у конструктор класу *Wayroutes* у вигляді чотирьох аргументів. Після цього, створюється нова сутність типу *Wayroutes*, що додається до списку *QList<Wayroutes> foo*. В кінці алгоритму метод *LoadData* повертає контейнер *QList<Wayroutes> foo*.

Метод *UpdateDirections* класу *Wayroutes* виконує функцію оновлення змінних *firstSt* та *lastSt* у сутності типу *Routes*, встановлюючи першу та останню станцію певного рейсу, тим самим формуючи його напрямок. Дана функція приймає покажчик *dataRoutes* на об'єкт типу *Routes* та нічого не повертає. При її виклику спочатку проводиться пошук усіх зупинок, що належать даному рейсу

та занесення їх до окремого контейнеру *foo* типу *QList<Wayroutes>*. Після цього відбувається сортування списку *foo* за часом відправлення. Згодом, якщо список не пустий, змінним *firstSt* та *lastSt* об'єкту даного рейсу присвоюється перший та останній елемент списку відповідно.

У конструкторі класу *ListModel* реалізовано функцію створення нової сутності типу *ListModel*. Дана функція приймає наступні значення: покажчик на список *_m_data*, що містить усі сутності типу *Wayroutes – QList<Wayroutes>**; покажчик на список *_routes_data*, що містить усі сутності типу *Routes – QList<Routes>*;* покажчик на список *_stations_data*, що містить усі сутності типу *Stations – QList<Stations>*;* Функція не повертає жодних значень. При її виклику створюється нова сутність типу *ListModel*.

У конструкторі класу *routeMenuModel* реалізовано функцію створення нової сутності типу *routeMenuModel*. Дана функція приймає покажчик на список *_m_data*, що містить усі сутності типу *Routes – QList<Routes>*. Функція не повертає жодних значень. При її виклику створюється нова сутність типу *routeMenuModel*.

У конструкторі класу *stationMenuModel* реалізовано функцію створення нової сутності типу *stationMenuModel*. Дана функція приймає покажчик на список *_m_data*, що містить усі сутності типу *Stations – QList<Stations>*. Функція не повертає жодних значень. При її виклику створюється нова сутність типу *stationMenuModel*.

3.3.2. Модуль пошуку рейсу за його номером

Модуль пошуку рейсу за його номером створений для обробки даних, що були введені користувачем, та формування даних, що мають бути виведені на екрані мобільного пристрою користувача. Даний модуль реалізовано в наступних класах: *ListModel* та *routeMenuModel*.

Метод *data* у класі *ListModel* є автоматично згенерованим та виконує функцію формування різних строк, основуючись на певній ролі для виводу їх у списку зупинок рейсів у графічному інтерфейсі. Дана функція приймає наступні

значення: змінну індексу зупинки *index* типу *QModelIndex*, що потрібно вивести та змінну ролі *role* цілочисельного типу *int*. Вона повертає змінну типу *QVariant*, що містить сформовану строку. При виклику даного методу, за допомогою оператору *switch* програма формує певну строку, базуючись на отриманій змінній ролі *role*. Метод оброблює 3 ролі: *Route, wayRoute* та *time*. Для ролі *Route* функція повертає строку, що містить в собі номер рейсу для потрібної зупинки. Для ролі *wayRoute* метод повертає строку із назвою станції для потрібної зупинки. Для ролі *time* функція повертає строку формату “час прибуття потягу на станцію-час відправлення потягу від станції”.

Перевантажений метод *specify* у класі *ListModel* виконує функцію пошуку рейсу за введеним користувачем номеру рейсу. Дана функція приймає змінну *rNumber* строкового типу – *QString* та повертає змінну логічного типу – *bool*. При її виклику програма очищує список *m_data* для того, щоб записати нові дані, що будуть виводитися користувачеві на екран. Після цього програма перетворює номер рейсу із строкового типу даних *QString* у цілочисельний тип *int* та присвоює отримане значення у змінну *routeNumber*, що має цілочисельний тип *int*. Після цього програма шукає сутність типу *Routes* з номером рейсу, що відповідає значенню змінної *routeNumber*. Якщо такої сутності не було знайдено, функція повертає значення *false*. Згодом проходить перебір усіх зупинок типу *Wayroutes*, і, якщо номер рейсу змінної *routeNumber* співпадає із номером рейсу, що відповідає поточній зупинці, то дана зупинка додається до списку *m_data*. Після даного циклу функція повертає значення *true*.

Метод *specifySearch* у класі *routeMenuModel* виконує функцію пошуку рейсів за номером, що вводить користувач у поле вводу на сторінці пошуку рейсу за його номером та повернення списку із сутностей типу *Routes* для подальшого виведення даних, що містяться у ньому, у вигляді випадаючого списку. Дана функція приймає змінну *rNumber* строкового типу – *QString* та повертає змінну логічного типу – *bool*. При її виклику програма очищує список *m_data* для того, щоб записати нові дані, що будуть виводитися користувачеві на екран. Після цього програма перебирає усі об’єкти типу *Routes* у списку *routes_data*. У циклі йде процес пошуку рейсів, номер яких містить введені

користувачем цифри. Якщо такі рейси були знайдені, програма додає їх до списку *m_data*. Потім, якщо список *m_data* не є пустим, функція повертає змінну логічного типу *bool* зі значенням *true*, інакше вона повертає значення *false*.

Метод *getByName* класу *routesMenuModel* виконує функцію отримання номеру рейсу зі змінної строкового типу *QString*, що передається у вигляді аргументу у функцію та має формат “номер рейсу, назва початкової станції-назва кінцевої станції”. Метод приймає змінну *_index* строкового типу *QString* та повертає змінну строкового типу *QString*. При викликанні даної функції програма розділяє отриману змінну *_index* за знаком “,” на дві окремі строки. Після цього повертає першу строку з розділеної змінної та повертає її значення у строковому вигляді.

Метод *getRouteStation* у класі *routeMenuModel* виконує функцію формування строки із напрямком рейсу у вигляді “назва початкової станції-назва кінцевої станції”. Даний рядок виводиться у випадаючому списку разом із номером рейсу, коли користувач вводить номер рейсу у поле вводу на сторінці пошуку рейсів за номером. Дана функція приймає змінну *_index* строкового типу – *QString* та повертає змінну строкового виду – *QString*. При її виклику програма програма перебирає усі об’єкти типу *Routes* у списку *routes_data*. У циклі йде процес пошуку рейсів, номер яких співпадає з отриманим у вигляді аргументу номером. Якщо такі рейси були знайдені, програма повертає змінну строкового типу *QString*, що містить напрямок певного рейсу.

3.3.3. Модуль пошуку рейсів за напрямком

Модуль пошуку рейсів за напрямком створений для обробки даних, що були введені користувачем та формування даних, що мають бути виведені на екрані мобільного пристрою користувача. Даний модуль реалізовано в наступних класах: *ListModel* та *stationMenuModel*.

Метод *data* у класі *ListModel* є автоматично згенерованим та виконує функцію формування різних строк, основуючись на певній ролі для виводу їх у списку зупинок рейсів у графічному інтерфейсі. Дана функція приймає наступні

значення: змінну індексу зупинки *index* типу *QModelIndex*, що потрібно вивести та змінну ролі *role* цілочисельного типу *int*. Вона повертає змінну типу *QVariant*, що містить сформовану строку. При виклику даного методу, за допомогою оператору *switch* програма формує певну строку, базуючись на отриманій змінній ролі *role*. Метод оброблює 3 ролі: *Route*, *wayRoute* та *time*. Для ролі *Route* функція повертає строку, що містить в собі номер рейсу для потрібної зупинки. Для ролі *wayRoute* метод повертає строку із назвою станції для потрібної зупинки. Для ролі time функція повертає строку формату “час прибуття потягу на станцію-час відправлення потягу від станції”.

Перевантажений метод *specify* у класі *ListModel* виконує функцію пошуку рейсів за введеним користувачем напрямком. Дана функція приймає наступні змінні: назву першої станції *_firstStation* строкового типу – *QString*, назву останньої станції *_lastStation* строкового типу – *QString* та повертає змінну логічного типу – *bool*. При її виклику програма очищує список *m_data* для того, щоб записати нові дані, що будуть виводитися користувачеві на екран. Після цього програма починає пошук отриманих станцій. Якщо хоча б одна станція не була знайдена – функція повертає логічне значення типу *bool* – *false*. Згодом програма створює контейнер *tempHash* типу *QMultiHash<int, Wayroutes>*, що буде зберігати кожну станцію та номер рейсу, якому відповідає та чи інша станція. Потім програма за допомогою циклу додає до контейнеру *tempHash* усі об’єкти типу *Wayroutes* із ключами у вигляді номеру рейсу. Потім програма починає перебирати всі рейси за допомогою циклу. Для кожного рейсу перебираються усі їх зупинки, та, якщо назви станцій із отриманих змінних *_firstStation* та *_lastStation* були знайдені у даному рейсі, вона додає зупинки цього рейсу до списку *m_data* типу *QList<Wayroutes>*, попередньо відсортувавши їх за датою та часом відправлення від станції для коректного відображення. Якщо у результаті виконання даного алгоритму список *m_data* не пустий, функція повертає значення *true*, інакше вона повертає значення *false*.

Метод *specifySearch* у класі *stationMenuModel* виконує функцію пошуку станцій базуючись на тому, що вводить користувач у поле вводу на сторінці пошуку рейсу за його напрямком та повернення списку із сутностей типу

Stations для подальшого виведення даних, що містяться у ньому, у вигляді випадаючого списку. Дана функція приймає змінну *stationName* строкового типу – *QString* та повертає змінну логічного типу – *bool*. При її виклику програма очищує список *m_data* для того, щоб записати нові дані, що будуть виводитися користувачеві на екран. Після цього програма перебирає усі об'єкти типу *Stations* у списку *stations_data*. У циклі йде процес пошуку станцій, назва яких містить введену користувачем строку. Якщо такі рейси були знайдені, програма додає їх до списку *m_data*. Потім, якщо список *m_data* не є пустим, функція повертає змінну логічного типу *bool* зі значенням *true*, інакше вона повертає значення *false.*

3.3.4. Модуль пошуку рейсу за станцією

Модуль пошуку рейсів за станцією створений для обробки даних, що були введені користувачем та формування даних, що мають бути виведені на екрані мобільного пристрою користувача. Даний модуль реалізовано в наступних класах: *ListModel* та *stationMenuModel*.

Метод *data* у класі *ListModel* є автоматично згенерованим та виконує функцію формування різних строк, основуючись на певній ролі для виводу їх у списку зупинок рейсів у графічному інтерфейсі. Дана функція приймає наступні значення: змінну індексу зупинки *index* типу *QModelIndex*, що потрібно вивести та змінну ролі *role* цілочисельного типу int. Вона повертає змінну типу *QVariant*, що містить сформовану строку. При виклику даного методу, за допомогою оператору *switch* програма формує певну строку, базуючись на отриманій змінній ролі *role*. Метод оброблює 3 ролі: *Route, wayRoute* та *time*. Для ролі *Route* функція повертає строку, що містить в собі номер рейсу для потрібної зупинки. Для ролі *wayRoute* метод повертає строку із назвою станції для потрібної зупинки. Для ролі *time* функція повертає строку формату "час прибуття потягу на станцію-час відправлення потягу від станції".

Перевантажений метод *specify* у класі *ListModel* виконує функцію пошуку рейсів за введеною користувачем. Дана функція приймає змінну назви станції

_firstStation строкового типу – *QString* та повертає змінну логічного типу – *bool*. При її виклику програма очищує список *m_data* для того, щоб записати нові дані, що будуть виводитися користувачеві на екран. Після цього програма починає пошук отриманих станцій. Якщо хоча б одна станція не була знайдена – функція повертає логічне значення типу *bool* – *false*. Згодом програма створює контейнер *tempHash* типу *QMultiHash<int, Wayroutes>*, що буде зберігати кожну станцію та номер рейсу, якому відповідає та чи інша станція. Потім програма за допомогою циклу додає до контейнеру *tempHash* усі об'єкти типу *Wayroutes* із ключами у вигляді номеру рейсу. Потім програма починає перебирати всі рейси за допомогою циклу. Для кожного рейсу перебираються усі їх зупинки, та, якщо назва станції з отриманої змінної *_firstStation* була знайдена у даному рейсі, вона додає зупинки цього рейсу до списку *m_data* типу *QList<Wayroutes>*, попередньо відсортувавши їх за датою та часом відправлення від станції для коректного відображення. Якщо у результаті виконання даного алгоритму список *m_data* не пустий, функція повертає значення *true*, інакше вона повертає значення *false*.

3.4. Висновки до розділу

У даному розділі було описано склад файлів розробленого проекту, перераховано папки та файли, що були створені під час розробки мобільного додатку.

Розробка програмного засобу велася в середовищі *QtCreator* із шаблоном «Приложение *Qt Quick* - пустое». Назва розробленого проекту – *train_schedule*. Усі файли проекту розташовані в папці *C:\ projecTtrainSchedule\train_schedule*.

Папка *train_schedule* містить автоматично створені файли: *train_schedule.pro, qml.qrc, main.qml* та *main.cpp*. У ході розробки проекту були створені наступні файли: *listmodel.cpp, listmodel.h, routemenumodel.cpp, routemenumodel.h, routes.cpp, routes.h, stationmenumodel.cpp, stationmenumodel.h, stations.cpp, stations.h, wayroutes.cpp, wayroutes.h, listRouteDisplay.qml,*

timetableDisplay.qml, stationDisplay.qml, welcomeScreen.qml, Routes.txt, Stations.txt, Wayroutes.txt.

Було детально описано модулі розробленого програмного засобу, а саме: модуль ініціалізації даних та завантаження даних, модуль пошуку рейсу за номером, модуль пошуку рейсів за напрямком та модуль пошуку рейсів за станцією. Було детально описано алгоритми даних модулів.

ВИСНОВКИ

Дипломний проект присвячений тематиці моніторингу розкладу руху залізничного транспорту. У ході виконання дипломного проекту було досліджено розробку крос-платформового додатку для мобільних операційних систем *Android* та *iOS*.

В першому розділі дипломного проекту було розглянуто аналоги програмних засобів для мобільних та стаціонарних платформ для моніторингу розкладу руху залізничного транспорту. Було розглянуто засоби крос-платформової розробки додатків для мобільних операційних систем *Android* та *iOS*.

Для стаціонарних платформ було розглянуто сайти «*poezdato.net*» та офіційний сайт Укрзалізниці – «*uz.gov.ua*». Був зроблений короткий огляд даних сервісів та їх основних функцій.

Для мобільних платформ було розглянуто наступні програмні додатки: «Моя Електричка», «Розклади приміських поїздів України - *UZTrains*», «Яндекс.Электрички» та «Укрзалізниця». Був зроблений короткий огляд даних мобільних програмних засобів та їх основних функцій.

Було розглянуто наступні засоби крос-платформової розробки додатків для мобільних операційних систем *Android* та *iOS*: фреймворк *Qt,* фреймворк *Xamarin* та фреймворк *Ionic*. Були розглянути їх основні переваги та недоліки, їх принципи роботи. Було досліджено, у яких ситуаціях потрібно використовувати той чи інший засіб розробки для мобільних платформ: *Qt* підійде для розробки мобільних додатків, що потребують обробки великої кількості даних, *Xamarin* та *Ionic* підійде для розробки мобільних додатків, чий інтерфейс повинен бути максимально схожим на «рідний» для тої чи іншої мобільної оперативної системи.

У другому розділі дипломного проекту було спроектовано мобільний додаток для моніторингу розкладу руху залізничного транспорту. Було обрано мову програмування та платформу розробки, визначено основні функції програмного засобу та спроектовано основні класи програмного засобу.

Було обрано мову програмування та платформу розробки: *C++* із фреймворком *Qt*, мова програмування для розробки графічного інтерфейсу *QML*, середовище розробки *QtCreator* із засобом розробки мобільних додадтків *AndroidSDK*.

Основними функціями програмного засобу було визначено: пошук рейсу за його номером, пошук рейсів за напрямком та пошук рейсів за станцією. Для кожної функції було описано їх алгоритм роботи та їхню взаємодію з користувачем.

Було спроектовано наступні класи: *Routes, Stations, Wayroutes, ListModel*, *routeMenuModel* та *stationMenuModel*. Були визначенні їх змінні та методи. Було описано, які типи та назви мають змінні класів, які аргументи приймає кожна функція та які значення вона повертає.

У третьому розділі дипломного проекту було описано склад файлів розробленого проекту, перераховано папки та файли, що були автоматично створені та створені під час розробки мобільного додатку.

Розробка програмного засобу велася в середовищі *QtCreator* із шаблоном «Приложение *Qt Quick* - пустое». Назва розробленого проекту – *train_schedule*. Усі файли проекту розташовані в папці *C:\ projecTtrainSchedule\train_schedule*.

Папка *train_schedule* містить автоматично створені файли: *train_schedule.pro, qml.qrc, main.qml* та *main.cpp*. У ході розробки проекту були створені наступні файли: *listmodel.cpp, listmodel.h, routemenumodel.cpp, routemenumodel.h, routes.cpp, routes.h, stationmenumodel.cpp, stationmenumodel.h, stations.cpp, stations.h, wayroutes.cpp, wayroutes.h, listRouteDisplay.qml, timetableDisplay.qml, stationDisplay.qml, welcomeScreen.qml, Routes.txt, Stations.txt, Wayroutes.txt.*

Було детально описано модулі розробленого програмного засобу, а саме: модуль ініціалізації даних та завантаження даних, модуль пошуку рейсу за номером, модуль пошуку рейсів за напрямком та модуль пошуку рейсів за станцією. Було детально описано алгоритми даних модулів.

У звіті дипломного проекту були описані основні етапи проектування та розробки мобільного додатку для моніторингу розкладу руху залізничного транспорту.

Дипломний проект було виконано з дотриманням діючих стандартів та положень [14], [15].

СПИСОК БІБЛІОГРАФІЧНИХ ПОСИЛАНЬ ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Р. Лафоре. Объектно-ориентированное программирование в *C*++. 4-е изд. – Санкт-Петербург: Питер, 2004. – 924 с.
2. Сигналы и слоты [електронний ресурс] – Режим доступу: *http://doc.crossplatform.ru/qt/4.6.x/signalsandslots.html* (дата звернення 15.05.2021)
3. М. Шлее. *Qt* 5.10. Профессиональное программирование на *C*++. Санкт-Петербург: БХВ-Петербург, 2018. – 1052 с.
4. Достоинства и недостатки *Xamarin* [електронний ресурс] – Режим доступу: *https://habr.com/ru/company/microsoft/blog/415833/* (дата звернення 15.05.2021)
5. *Ionic framework*. Обзор экосистемы [електронний ресурс] – Режим доступу: *https://habr.com/ru/company/microsoft/blog/415833/* (дата звернення 15.05.2021)
6. Майкл С. Миковски, Джош К. Пауэлл. Разработка одностраничных веб-приложений. – Москва: ДМК Пресс, 2014. – 512 с.
7. Введение – *Apache Cordova* [електронний ресурс] – Режим доступу: *https://cordova.apache.org/docs/ru/dev/guide/overview/* (дата звернення 15.05.2021)
8. Фримен А. *Angular* для профессионалов. – СПб.: Питер, 2018. – 800 с.
9. Плюсы и минусы разработки приложений на *Ionic* [електронний ресурс] - Режим доступу: *https://dou.ua/lenta/articles/ionic-development/* (дата звернення 15.05.2021)
10. С. Бьярне. Программирование: принципы и практика с использованием *C*++, 2-е изд. : Пер. с англ. – Москва: Вильямс, 2016. – 1 328 с.
11. Филлипс Б., Стюарт К., Марсикано К. *Android*. Программирование для профессионалов. 3-е изд. – Санкт-Петербург: Питер, 2017. – 688 с.
12. *Qt Creator - A Cross-platform IDE for Application Development* [електронний ресурс] – Режим доступу: *https://www.qt.io/product/development-tools* (дата звернення 15.05.2021)
13. Жасмин Бланшет, Марк Саммерфилд. *QT4* программирование *GUI* на *C*++. – Москва: КУДИЦ-ПРЕСС, 2008. – 738 с.

14. ДСТУ 3008-95. Документація. Звіти у сфері науки і техніки. Структура і правила оформлення / Держстандарт України. – Вид. офіц. – [Чинний від 1995-02-23]. – Київ, 2007. – 86с.
15. Слободян О. Положення про дипломні роботи (проекти) випускників Національного авіаційного університету. – К.: Видавництво НАУ, 2017. – 63с.